Geometrical and Logical Modelling of Cartographic Objects

Master Thesis in Computational Engineering

submitted by

Duane Mari I. Deang

born on 7 May 1975 in Legaspi, Philippines

written at

Institut für Informatik
Lehrstuhl für Künstliche Intelligenz
Friedrich-Alexander-Universität Erlangen-Nürnberg

Supervisor:	Prof. Dr. Günther Görz
Start of Work:	30 March 2000
End of Work:	2 October 2000

Ich versichere, daß ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und daß die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Erlangen, 29 September 2000

Summary

The oldest globe of the earth in existence is the Behaim globe (1492), which is kept at the Germanisches Nationalmuseum in Nuernberg. A multimedia information system of the globe has been developed at the Artificial Intelligence Department of the University of Erlangen-Nuernberg in cooperation with FORWISS and the Germanisches Nationalmuseum.

This work offers a beginning for the geometric modelling of cartographic objects and their topological and orientation relations in consideration of cognitive ideas about space and spatial relations. Enhancing systems such as the Behaim system with a geometrical model allows users to ask complex questions with a geographic context. The model is implemented using the description logic system CLASSIC.

Kurzfassung

Der älteste existierende Erdglobus, heute in den Sammlungen des Germanischen Nationalmuseums in Nürnberg, ist der Behaim-Globus (1492). Ein multimediales Informationssystem des Globuses wurde von dem Lehrstuhl für Künstliche Intelligenz an der Universität Erlangen-Nürnberg in Zusammenarbeit mit FORWISS und mit dem Germanischen Nationalmuseum entwickelt.

Diese Arbeit ist ein Ansatz für die geometrische Modellierung von kartographischen Objekten und ihren topologischen und orientierungsspezifische Relationen unter Berücksichtigung kognitiven Ideen über Raum und räumliche Relationen. Verbesserung von Systemen so wie das Behaim System mit einem geometrischen Modell bietet Benutzern die Möglichkeit komplizierte Anfragen in einem geographischen Kontext zu stellen. Das Modell ist mit dem Beschreibungslogiksystem CLASSIC implementiert.

Contents

Chapter 1

Introduction

1.1 Martin Behaim's Globe

In 1492, the merchant Martin Behaim initiated the creation of a globe of the earth. Today this masterpiece of European Renaissance art is the oldest existing globe and is kept at the Germanisches Nationalmuseum in Nuernberg. The globe shows a pre-columbian image of the earth and depicts a large number of small miniatures and texts dealing with various aspects of the different countries such as the inhabitants, plants and animals, as well as trading and religious issues. This makes the Behaim globe one of the most valuable pieces of art and history.

In celebration of the 500th anniversary of the globe¸there was a big museum exhibition in 1992 and a two-volume catalog was published¸which showcased the first results of research activities began in 1987. Among other scientific activities¸a complete photographic record of the globe was made. Based on the rich image and text materials gathered from these research activities¸a project was started between FORWISS and the Germanisches Nationalmuseum in Nuernberg to build a multimedia information system. The system functions as an electronic encyclopedia and presents comprehensive information about the globe and its background. It has two components: a graphical component to depict images of the globe¸and a text component to provide in-depth information about the globe. A hypermedia structure is used to pay particular attention to the various ways texts and images relate. Furthermore¸ a conceptual model of the globe was developed and linked to this structure to enable more complex queries.

1.2 Goal of the Thesis

The primary aim of this thesis is to develop a starting point for the geometric modelling of cartographic objects and their topological and orientation relations. This is done in consideration of cognitive ideas about space and spatial relations. Possible ways of implementation using a logic formalism, namely description logic (DL), are also explored. Augmented with a geometrical model, systems such as the Behaim system described above would be able to answer even more complex queries with a geographic context.

Figure 1.1: Behaim globe

1.3 Structure of the Thesis

Section 2.1 gives an overview on Geographic Information Systems, in particular, the types of questions they try to answer and the theories on which future systems may be based. Section 2.2 introduces Qualitative Spatial Reasoning, an important field of study necessary for the design of intelligent GIS's. Section 2.3 summarizes the important features of description logic and CLASSIC, the DL system used for the implementation.

Chapter 3 presents the main part of this work. Section 3.1 discusses the previous work done on modelling the Behaim globe which includes the concept taxonomy and the aggregation model. Section 3.2 explains how the topological and orientation relations between the objects can be modelled. The implementation in CLASSIC and examples of how geographic queries can be done in the resulting model are also described here.

Chapter 4 contains a summary of the work and gives a perspective on future work on the area.

Chapter 2

Fundamentals

2.1 Geographic Information Systems

As much as man is a slave of time, one cannot deny that he is also trapped in space. It is one of the things that constantly takes part in one's daily decisions, either directly or indirectly. It is therefore not surprising that man has developed tools to aid him in his spatial existence: maps since the early times, and, with the advent of computers, geographic information systems.

2.1.1 Geographic Queries

Basically, maps and geographic information systems have the primary purpose of recording knowledge about one's environment such that it can aid him or others at a later time. [25] suggests two primary classes of questions from users:

1. the "What's here" query which asks about information available about a particular location, and
2. the "Where's this" query which try to find out where certain phenomena occur.

Other terms used for these two classes are location-based queries [24] or *deiktische Frage* [22] for the first type and phenomenon-based queries [24] or *iterative Frage* [22] for the second one.

Another important type of spatial query, which may apply only to particular types of maps, is the so-called path or path-based query. Such queries depend on the existence of network structures such as roads although it can be argued that they can also be asked on "roadless" maps with an appropriate notion of connectivity.

Further typologies of spatial queries have been developed such as one based on the number of objects involved [Jelinek97] and the information provided by the user [25]. In [8]a formalization of spatial queries is offered based on three types of spatial entities (points, lines and regions) and three types of spatial relations (topological, set-theoretic and metric). Depending on the kind of relation between the query object and the entities in the search space, they have the following classification of queries:

- topological queries based on
  - adjacency relations
  - boundary relations

  - co-boundary relations

- set-theoretic queries based on
  - coincidence relations
  - element-of relations
  - containment relations
  - intersection relations

- metric queries based on
  - min/max relations
  - range relations

2.1.2 Naive Geography

Most Geographic Information Systems built to answer such queries need extensive training. The users need to familiarize themselves with the system terminology and to understand the formalizations used in representing geographic data and how to get the information they need.

In [12] a field of study that is concerned with formal models of the common-sense geographic world is presented. It is called Naive Geography and is defined as "the body of knowledge that people have about the surrounding geographic world." Its aim is to compile a set of theories upon which future intelligent Geographic Information Systems will be based. These new systems are designed to follow human intuition and are, therefore, useful for a wider range of people.

Following are a few of Egenhofer and Mark's interesting observations which they believe may contribute to a Naive Geography:

- Naive geographic space is two-dimensional.
- The earth is flat.
- People use multiple conceptualizations of geographic space.
- Geographic space has multiple levels of detail.
- Boundaries are sometimes entities, sometimes not.
- Topology matters, metric refiines.
- People have biases toward north-south and east-west directions.
- Distances are assymetric.

2.2 Qualitative Spatial Reasoning

In their paper, Egenhofer and Mark also emphasized the importance of qualitative reasoning methods to the development of systems based on Naive Geography. In qualitative reasoning a variable may only take a small, predetermined number of values. For example, instead of using numbers one may just use the values positive, zero or negative. Qualitative reasoning allows inferences with incomplete information, but it does not do so by using probabilistic or fuzzy techniques. This is achieved by not differentiating between quantities unless there is enough evidence to do so. "Indistinguishable" values are collapsed into one equivalence class which becomes a qualitative value. Qualitative reasoning about space involves many aspects. Some of them are described shortly below.

2.2.1 Basic Elements

Traditionally, in mathematical theories of space, points are considered the basic elements out of which more complex spatial objects are composed. Points are abstract entities with no physical extension nor mass. For this reason, researchers in Qualitative Spatial Reasoning tend to not use points as the primitive spatial entity. Most prefer to take regions of space as basic elements. Afterall, it seems strange to think that regions are composed of dimensionless points.

2.2.2 Topology

In topology, regions are the preferred basic entities. Topology, also informally called "rubber sheet geometry", describes properties of a spatial object that do not vary depending on scale or orientation.

Figure 2.1 shows the 8 topological relations that can hold between two regions. These 8 relations based on a notion of connectedness form a jointly exhaustive and pairwise disjoint (JEPD) set and is often referred to as the RCC8 relations.

2.2.3 Orientation

Orientation relations require three elements: a primary object, a reference object and a frame of reference. Three types of frames of reference are distinguished:

1. An extrinsic frame of reference requires an external, unchanging orientation, such as a fixed coordinate system.
2. A deictic frame of reference involves the speaker or observer.
3. An intrinsic frame of reference uses a property of the reference object, such as the "front".

Orientation relations, that seemingly have only a reference object and a primary object, use an implicit extrinsic frame of reference (for example, the cardinal compass directions).

2.2.4 Distance and Size

Distance and size relations are divided into two main groups: those that use an "absolute" scale, and those that use a kind of relative measurement. Absolute relations simply make use

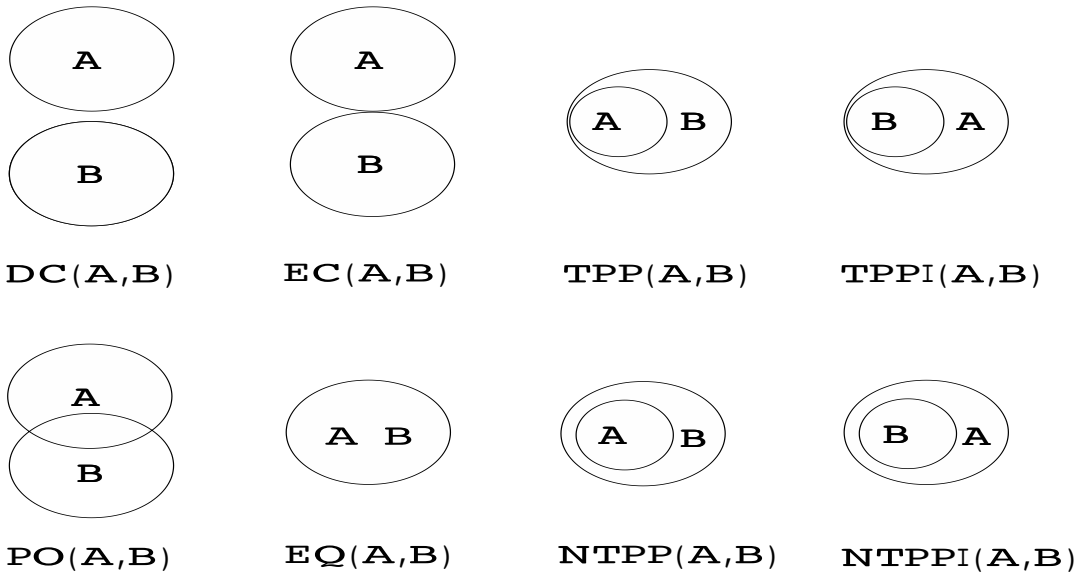


Figure 2.1: Basic topological relations in the RCC8 theory

of qualitative measures such as near or far, or big or small. Relative measures, on the other hand, allow an object to be described as being much larger or much farther than another.

2.2.5 Shape

Descriptions of shape are done by either constraining the possible shapes of a region, or constructing more complex shapes from simpler ones, like in constructive solid geometry. Some properties exploited in describing an objects shape are: whether it has holes, whether it is in one piece or more, and convexity and containment properties.

2.3 Description Logic

Description Logic (DL) is a structuring formalism for knowledge representation that has its roots from Semantic Networks and Frames. A Description Logic system uses a concept language, a set of constructs, to denote classes of individuals and relationships among them. A concept language is composed of symbols representing Concept Names, Role Names and Individual Names plus a number of constructors that allow the formation of concept expressions and role expressions.

A DL knowledge base has two components:

1. an intensional component called the T-Box, which gives a general schema concerning the classes of individuals and their properties and relationships, and
2. an extensional component called the A-Box, which is a (partial) instantiation of the schema.

Constructor Name	Syntax	Semantics
concept name	A	$A^I \subseteq \triangle^I$
top	$\top$	$\triangle^I$
bottom	$\bot$	$\emptyset$
conjunction	C$\sqcap$D	$C^I \cap D^I$
disjunction	C$\sqcup$D	$C^I \cup D^I$
negation	$\neg$C	$\triangle^I \setminus C^I$
universal quantification	$\forall$R.C	$\{d_1 \mid \forall d_2\colon\ (d_1,d_2) \in R^I \rightarrow d_2 \in C^I\}$
existential quantification	$\exists$R.C	$\{d_1 \mid \exists d_2\colon\ (d_1,d_2) \in R^I \wedge d_2 \in C^I\}$
number restrictions	($\geq$n R)	$\{d_1 \mid \sharp\{d_2 \mid (d_1,d_2) \in R^I\} \geq \mathrm{n}\}$
	($\leq$n R)	$\{d_1 \mid \sharp\{d_2 \mid (d_1,d_2) \in R^I\} \leq \mathrm{n}\}$
collection of individuals	$\{a_1,\ldots,a_n\}$	$\{a_1^I,\ldots,a_n^I\}$
role name	P	$\mathrm{P}^I \subseteq \triangle^I \times \triangle^I$
role conjunction	Q$\sqcap$R	$Q^I \cap R^I$

Table 2.1: Syntax and Semantics of concept language constructors

Table 2.1 summarizes the concept language constructors. A and B denote concept names, P role names, a and b individual names. Concept expressions are denoted with C, D and role expressions with Q, R. There are a number of variants of description logics that differ in the constructors they allow.

The DL system chosen for the implementation part of this thesis is the CLASSIC ("CLAS-Sification of Individuals and Concepts") implementation in Common LISP. CLASSIC provides only a subset of the constructors shown in Table 2.1 (for example, disjunction and negation are not supported) and uses an incomplete structural subsumption algorithm instead of a full tableaux calculus procedure. Nevertheless, this limited expressivity can be overcome in some cases and is enough to be able to construct useful geographic queries, as will be shown in the next chapter.

Concepts and individuals in CLASSIC come either from the CLASSIC or the HOST realm. CLASSIC concepts represent classes of individuals from the domain being modeled while HOST concepts describe individuals from the implementation language (in this case LISP), such as numbers, strings and lists.

The CLASSIC system provides the following reasoning services:

- Completion: Logical consequences of assertions about individuals and descriptions of concepts are computed through inheritance, combination or propagation inference mechanisms. CLASSIC also detects contradictory facts that may be accidentally asserted on individuals or concepts.
- Classification and subsumption: Concepts and individuals are automatically put into a hierarchy based on generality/specifity.
- Rule application: CLASSIC allows simple forward-chaining rules with concepts as antecedents and consequents.

The real power of CLASSIC, though, lies in its extensibility. The two operators TESTC (for CLASSIC concepts) and TESTH (for HOST concepts) allow user-defined procedures to be used in describing concepts.

Chapter 3

Modelling Maps with Description Logic

3.1 Phenomenological Model

The first step in modelling maps is deciding which types of objects are important to be represented and what their properties are. In the case of the Behaim globe, this has already been done in [33]. The concepts have also been classified under a concept taxonomy (is-a relation) and an aggregation hierarchy (part-whole relation). The concept taxonomy of globe objects are represented in Figure 3.1 and the aggregation hierarchy in Table 3.1.

The Behaim globe conceptual model has already been implemented in YAK, a description logic system, and in CLOS. Several segments of the Behaim globe have already been encoded using the model. Apart from numerous object properties, the NG-Region also has the role nongeo-teil-von (nongeographic part of) while the G-Region has the roles geo-teil-von (geographic part of), geo-teile (geographic parts), nongeo-teile (nongeographic parts) and geo-nachbar (geographic neighbor). The values of the relations were all determined manually.

The conceptual model was partially implemented in CLASSIC based on the CLOS model (See Appendix). The CLASSIC implementation uses a slightly different taxonomy which takes advantage of CLASSIC's ability to define multiple disjoint classes of objects. This is illustrated in Figure 3.2.

The aggregation hierarchy could only be integrated into the model in a weaker form. CLASSIC does not have the constructors NOT and OR. Therefore, it is not possible define more specific value constraints like in the following manner:

```
(cl-define-disjoint-primitive-concept 'SEE '(and ZGR-WASSER
(ALL geo-teile (OR INSEL INSELGRUPPE STROEMUNG)))
'zgr-wassertype)
```

One has to be content with a more general constraint higher up in the concept taxonomy:

```
(cl-define-disjoint-primitive-concept 'G-REGION '(and GLOBUS-REGION
(ALL geo-teil-von GLOBUS-REGION)
(ALL geo-nachbar GLOBUS-REGION)
(ALL geo-teile GLOBUS-REGION)
(ALL nongeo-teile NG-REGION)) 'ng-g-region)
```

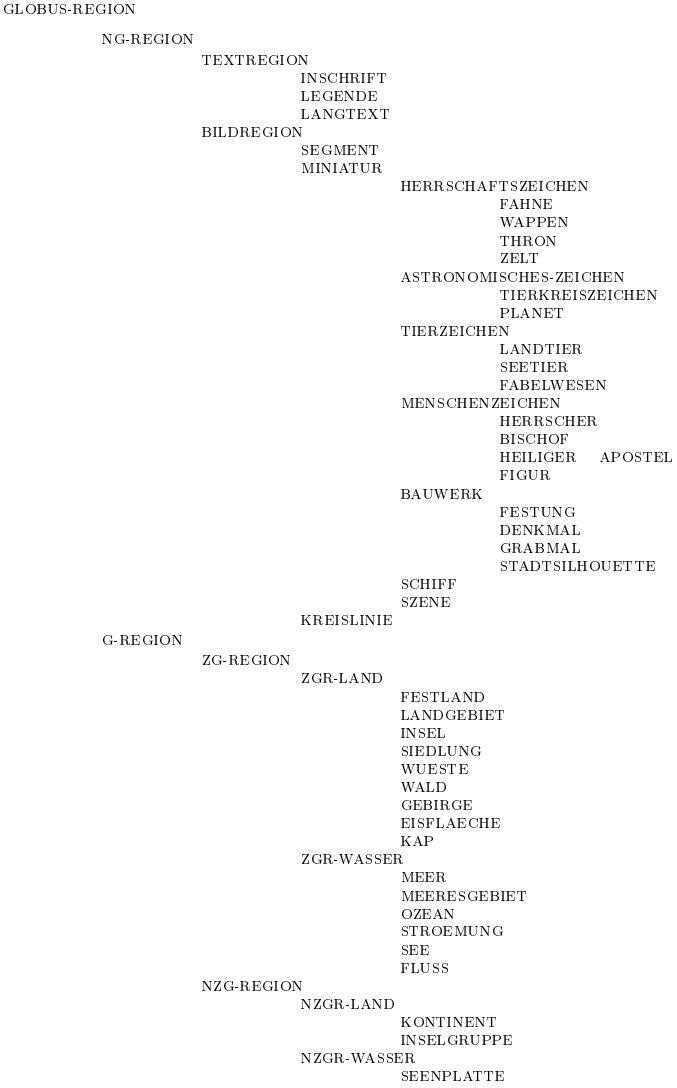


Figure 3.1: Partial concept taxonomy starting from Globus Region

G-Region	Possible Has-Part Values
Festland	all ZGR-Land and ZGR-Wasser except Festland, Meer
Landgebiet	-
Insel	all ZGR-Land and ZGR-Wasser except Insel, Meer
Wald	-
Gebirge	-
Kap	-
Eisflaeche	-
Stadt	-
Meer	Meeresgebiet, Insel, Inselgruppe, Stroemung
Meeresgebiet	-
See	Insel, Inselgruppe, Stroemung
Fluss	-
Stroemung	-
Kontinent	all ZGR-Land plus See, Fluss, Stroemung
Inselgruppe	Insel
Seenplatte	See

Table 3.1: Aggregation Hierarchy

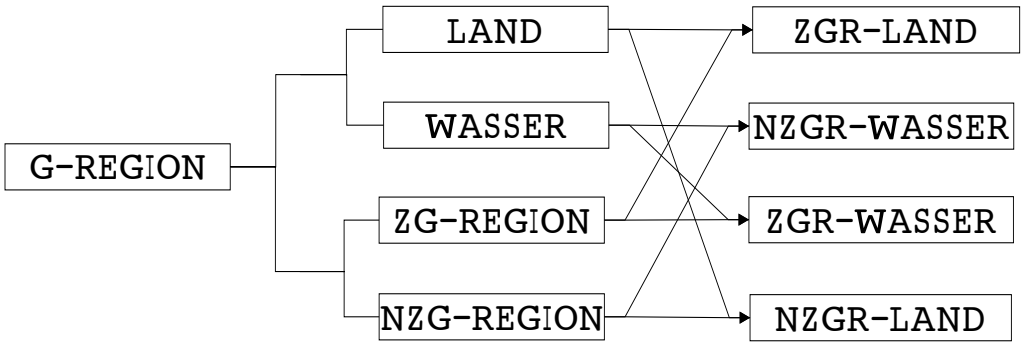


Figure 3.2: Alternative Taxonomy for G-Region

```
(cl-add-rule 'g-region-geo-teil-von @c{G-REGION}
             '(ALL geo-teil-von G-REGION))
(cl-add-rule 'g-region-geo-nachbar @c{G-REGION}
             '(ALL geo-nachbar G-REGION))
(cl-add-rule 'g-region-geo-teile @c{G-REGION}
             '(ALL geo-teile G-REGION))
```

The three rules are necessary in order to avoid circular definitions which are not permitted in CLASSIC.

The relation geo-teile was defined as the inverse of geo-teil-von and nongeo-teile the inverse of nongeo-teil-von. Furthermore, geo-teil-von and nongeo-teil-von were defined as attributes,i.e. they can assume a maximum of one value. In trying out data from segment 11, CLASSIC complained of errors such as the following:

```
*CLASSIC ERROR* while processing
(CL-CREATE-IND STADT_GENUA_11_3
(AND STADT (FILLS GEO-TEIL-VON LANDGEBIET_ITALIA_11_3)
(FILLS GEO-NACHBAR STADT_FLORENTS_11_3)
(FILLS NONGEO-TEILE INSCHRIFT_GENUA_11_3)))
occurred on object @i{INSCHRIFT_GENUA_11_3-*INCOHERENT*}
along role-path (@r{NONGEO-TEIL-VON}):
Inconsistent number restriction for role @r{NONGEO-TEIL-VON}
- at-least: 2; at-most: 1.
```

Upon inspection of the original CLOS implementation data, the reason turned out to be the following:

```
(make-instance ' STADT
:NAME          :stadt_genua_11_3
:NONGEO-TEILE    (list :inschrift_genua_11_3))
(make-instance ' INSCHRIFT
:NAME          :inschrift_genua_11_3
:NONGEO-TEIL-VON         :landgebiet_Italia_11_3)
(make-instance ' LANDGEBIET
:NAME          :landgebiet_Italia_11_3
:NONGEO-TEILE    (list :inschrift_Italia_11_3))
```

At some points, the part/whole relations nongeo-teile and nongeo-teil-von seem to be based on two different notions: one "political" and the other spatial.

3.2 Geometrical Model

The next step in modelling maps is describing how the objects are spatially related to one another. In [22] a spatial-query component was developed for the Behaim system in Lisp to compute answers to queries about distance and orientation. It could answer queries like "Name all SEETIER west of SCHIFF_12_3." and "Name all ZG-REGION near SCHIFF_12_3."

The next sections will describe how similar questions about topological and orientation relations can be answered using CLASSIC. All examples will refer to a simple map also based

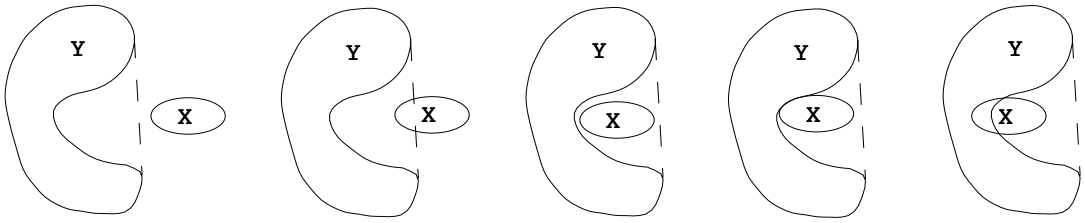


Figure 3.3: Some of the new possible distinctions with the addition of a convex hull primitive.

on the Behaim concept taxonomy but composed only of rectangular regions. (See Appendix B.) The computations are based on two attributes:

1. globus-koord: a list consisting of one coordinate-pair (latitude longitude) representing the center of the object, and

2. globus-polyon: a list of coordinate pairs enumerating the corners of the polygon that represents the region.

3.2.1 Topology

The key to qualitative representation is that the distinctions made between quantity spaces are relevant to the behaviour being modelled. Distinctions are only introduced if they are necessary to model a particular aspect of the domain with respect to the task at hand. [5] The usual eight topological relations in the RCC8 theory, for example, could be extended by using the notion of a convex hull into 23 JEPD relations. [6, 7] Figure 3.3 illustrates some of the new distinctions possible in the resulting RCC23 theory. The 23 relations can, in turn, be further refined into 47 relations and so on.

For modelling maps, the original 8 JEPD relations seem to be adequate, if not too much. Most people, for example, would not care about the distinction between NTPP and TPP. Both are just regarded as an inside relation.

To implement this in CLASSIC, the function get-rcc(obj1 obj2) is needed which returns the RCC relation between the objects. Since the 8 relations are JEPD relations, there can only be one relation that holds between any two regions.

The get-rcc function is then used to define procedures to be used in a TESTC expression. For example, the procedure is-ntpp?(ind ref) returns T if ind is a nontangential proper part of ref and nil otherwise.

To answer queries such as "What G-Regions are nontangential proper parts of continent K1?", one first needs to express the query as a concept:

```
(cl-define-concept 'query1 '(and G-REGION (TESTC is-ntpp? K1)))
```

Once the concept is defined, CLASSIC automatically classifies all instances that satisfy the description of the concept. The answer to the query can then be retrieved by asking for the instances of the concept:

```
(cl-concept-instances @query1)
(@i{W1} @i{S4} @i{S3} @i{S2})
```

The conjunction constructor can be used to define complexer queries like "What G-Regions are nontangential proper parts of continent K1 and partially overlaps landmass LG1?":

```
(cl-define-concept 'query2 '(and g-region (testc is-ntpp? K1)
                                          (testc is-po? LG1)))
(cl-concept-instances @query2)
(@i{W1})
```

In able to collapse the TPP and NTPP relations into a single inside relation, the disjunction constructor is needed, which is not available in CLASSIC. One can get around this limitation by defining an is-inside? procedure which ORs the the results of procedures is-tpp? and is-ntpp?:

```
(defun is-inside? (ind ref)
 (cond ((or (is-tpp? ind ref) (is-ntpp? ind ref)) T)
       (T nil)
       )
 )
```

This can then be used in a TESTC expression to ask, for example, "What G-Regions are inside continent K1?":

```
(cl-define-concept 'query3 '(and g-region (testc is-inside? K1)))
(cl-concept-instances @query3)
(@i{S2} @i{S3} @i{S4} @i{W1} @i{S1} @i{LG3} @i{LG2} @i{LG1})
```

3.2.2 Orientation

As with the case in topological relations, deciding how many distinct qualitative classes there will be is important. Table 3.2 shows several possible qualitative models for orientation that use the cardinal compass directions as frame of reference.

For the simple world example, the 4-direction model was chosen. The procedure lies-in-dir?(ind ref dir) which returns T if ind lies in the direction dir of the ref object, and nil otherwise. In modelling globes, this function also has to take the distance between the objects into consideration when distinguishing between east and west. For example, it is not correct to say that France lies to the east of Germany because doing so would take the longer distance around the world.

The lies-in-dir? procedure is then used in TESTC expressions to define queries. Following are examples:

```
; "Which G-Regions lie to the south of island I7?"
(cl-define-concept 'query4 '(and g-region
                             (testc lies-in-dir? I7 south)))
(cl-concept-instances @query4)
(@i{M1} @i{S13} @i{S8} @i{S7} @i{S6}
```

Model	Interval Values (in degrees)
4 Directions	N: (45, 135], W: (135, 225], S: (225, 315], E: (315, 45]
8 Directions	E: (337, 22], NE: (22, 67], N: (67,112], NW: (112, 157], W: (157, 202], SW: (202, 247], S: (247, 292], SE: (292, 337]
N/S	N: (0,180], S: (180, 0]
W/E	W: (90, 270], E: (270, 90]
Combination	NE: (0,90], NW: (90, 180] , SW: (180, 270], SE: (270, 0]

Table 3.2: Examples of qualitative models for orientation

```
 @i{S5} @i{I5} @i{I4} @i{I3} @i{I2} ...)

; "Which G-Regions lie to the south of island I7 and west of island I3?"
(cl-define-concept 'query5 '(and g-region
                             (testc lies-in-dir? I7 south)
                             (testc lies-in-dir? I3 west)))
(cl-concept-instances @query5)
(@i{K2} @i{F2} @i{LG5} @i{I2} @i{S7}
 @i{S8} @i{M1} @i{S6} @i{S5} @i{I1} ...)
```

By using opposite directions, it is possible to approximate the idea of betweenness:

```
; "Which G-Regions lie to the south of island I7 and north of island I2?
; (Which G-Regions lie between islands I7 and I2?)"
(cl-define-concept 'query6 '(and g-region
                             (testc lies-in-dir? I7 south)
                             (testc lies-in-dir? I2 north)))
(cl-concept-instances @query6)
(@i{IG1} @i{I1} @i{S5} @i{S6})
```

3.2.3 Scale

In maps, the notion of scale plays an important role. For example, the answer Erlangen to the question "What lies north of Africa?" although correct seems inappropriate. But Europe would be a "good" answer because it is on the same scale as Africa. Giving back Erlangen as answer just gives too much detail too soon, which the user may not wish.

The simplest solution to this problem is by querying only for objects that belong to the same class as the reference object:

```
; "Which islands lie to the south of island I7?"
```

G-Region	Scale
Kontinent, Ozean	very big
Festland, Meer	big
Landgebiet, Meeresgebiet, Inselgruppe	medium
Insel, Wueste, Gebirge, Eisflaeche, Seenplatte	small
Stadt, Wald, Kap, Stroemung, See, Fluss	very small

Table 3.3: Classification of G-Region according to scale

```
(cl-define-concept 'query7 '(and insel
                             (testc lies-in-dir? I7 south)))
(cl-concept-instances @query7)
(@i{I5} @i{I4} @i{I3} @i{I2} @i{I1})
```

This way, however, may be a little too constraining. If, for example, one asks "What lies west of Africa?", it would be reasonable to expect as answer not only South America but perhaps the Atlantic Ocean, as well. Reasoning like this can be integrated into the current model by introducing a new subclassification under the g-region object. Instances of regions belonging to the same scale would then be classified under the same subconcept. For the Behaim concept taxonomy, a 5-level scale is used. (Table 3.3)

The CLASSIC implementation is modified as follows:

- a new concept SCALED-G-REGION is defined under G-REGION:

```
(cl-define-primitive-concept 'SCALED-G-REGION 'G-REGION)
```

- the five classes of Scaled-G-Regions according to scale are defined as disjoint concepts:

```
(cl-define-disjoint-primitive-concept 'VERYBIG-REGION
                                      'SCALED-G-REGION 'scale)
(cl-define-disjoint-primitive-concept 'BIG-REGION
                                      'SCALED-G-REGION 'scale)
(cl-define-disjoint-primitive-concept 'MEDIUM-REGION
                                      'SCALED-G-REGION 'scale)
(cl-define-disjoint-primitive-concept 'SMALL-REGION
                                      'SCALED-G-REGION 'scale)
(cl-define-disjoint-primitive-concept 'VERYSMALL-REGION
                                      'SCALED-G-REGION 'scale)
```

- rules are added to automatically classify instances of the different G-Region types into their appropriate SCALED-G-REGION class:

```
(cl-add-rule 'kontinent-scale @KONTINENT 'VERYBIG-REGION)
(cl-add-rule 'ozean-scale @OZEAN 'VERYBIG-REGION)
(cl-add-rule 'festland-scale @FESTLAND 'BIG-REGION)....
```

Whenever instances of G-Region are declared, they are then automatically classified according to scale. And this scale information can be used to define preciser concepts for queries like "What is east of lake SE1?":

```
(cl-define-concept 'query8 '(and verysmall-region
                             (testc lies-in-dir? SE1 east)))
(cl-concept-instances @query8)
(@i{FL1} @i{S14} @i{S11} @i{S10} @i{S9} @i{S2} @i{S1})
```

3.2.4 Complex Queries

Forming complex queries based on both topological and orientation relations is straightforward using the AND constructor. Here are a few examples:

```
; "Which cities inside continent K1 lie west of forest W1?"
(cl-define-concept 'query9 '(and stadt
                             (testc is-inside? K1)
                             (testc lies-in-dir? W1 west)))
(cl-concept-instances @query9)
(@i{S2} @i{S1})

; "What are the eastern neighbors of sea M4?"
(cl-define-concept 'query10 '(and big-region
                              (testc is-ec? M4)
                              (testc lies-in-dir? M4 east)))
(cl-concept-instances @query10)
(@i{F2})

; "Which islands lying between island I11 and island group IG2
; are not part of island group IG1?"
(cl-define-concept 'query11 '(and insel
                              (testc lies-in-dir? I11 east)
                              (testc lies-in-dir? IG2 west)
                              (testc is-dc? IG1)))
(cl-concept-instances @query11)
(@i{I7} @i{I6} @i{I5})
```

Distance can be modelled qualitatively just as was done with orientation. Distinct distance classes can be defined based on intervals of quantitative values. For example, extremely near for objects within 10 units, very near for those within 11-30 units, near for those within 30-70 units, and so on. Then, as usual, procedures will be defined for these classes that can be used in TESTC expressions (is-extremely-near?, is-very-near?, etc.). Finally, they can be used in conjunction with the topological and orientation relations to define queries.

The problem with this approach, though, is to decide how many distance classes there should be and to which intervals they should map. Unlike in orientation where once the model type is decided, the interval that north maps to is clearly delineated, different people may define near differently. Indeed, even the partitioning scheme of intervals may vary. While some users may divide the intervals into partitions of equal length, others may decide to use a nonlinear scheme, such as the one described above.

A good alternative may be to not consider this problem at all and to just leave distance relations as quantitative relations. Users would then be free to define their own distance metrics. Therefore mixed qualitative-quantitative queries would be possible. In the CLASSIC

implementation, the only thing required is a procedure like lies-within?(ind ref dist) which returns T if ind is within dist distance from ref, and nil otherwise. Then, even more complicated queries like the following can be constructed:

```
; "Which cities inside continent K3 lie within 50 units
; east of lake SE1?"
(cl-define-concept 'query12 '(and stadt
                        (testc is-inside? K3)
                        (testc lies-in-dir? SE1 east)
                        (testc lies-within-dist? SE1 50)))
(cl-concept-instances @query12)
(@i{S9})
```

3.2.5 Limitations

One limitation in the CLASSIC implementation is the absence of the disjunction and negation constructors. Unfortunately, this is unavoidable when one needs to use OR or NOT on concepts, as would have been useful in implementing the aggregation constraints. However, when disjunction or negation is needed on TESTC expressions, there exists a way to do it. The trick is to perform the disjunction or negation outside of CLASSIC through a Lisp function which is then used in the TESTC expression. This was shown in defining the is-inside? procedure which simply returns the OR of the results of is-ntpp? and is-tpp?

Another limitation is that variable reference objects cannot be used in the TESTC expressions. All reference objects must be specified in advance. There is no way, for example, to formulate "Which cities are inside an island?" as a concept for querying. Such questions have to be broken down into a series of queries. Thus, for the example one has to first ask what the instances of the concept island are. And then for each of the island instances, a query for the cities inside it has to be constructed.

3.2.6 Explicitly Storing Relations

Relations are usually represented as roles in CLASSIC. Ideally, the spatial relations would be defined as roles and described perhaps with the help of user-defined functions. CLASSIC would then proceed to fill in the roles with instances that satisfy the role definitions. For example, orientation relations may be defined as:

```
(cl-define-primitive-role 'south :inverse 'north)
(cl-define-primitive-role 'east :inverse 'west)
```

And these roles would then be part of the globus-objekt definition like this:

```
(cl-define-primitive-concept 'GLOBUS-OBJEKT '(and GLOBUS-ROOT
(ALL globus-koord LIST)
(ALL globus-polygon LIST)
(ALL south (and GLOBUS-ROOT
           (TESTC lies-in-dir? ref south)))
(ALL north (and GLOBUS-ROOT
           (TESTC lies-in-dir? ref north)))
```

```
(ALL east (and GLOBUS-ROOT
          (TESTC lies-in-dir? ref east)))
(ALL west (and GLOBUS-ROOT
          (TESTC lies-in-dir? ref west)))))
```

Unfortunately, this is not possible because CLASSIC has no way of knowing what the reference object (ref) is. One possible solution to this is the following:

1. Create all instances of globus-objekt.
2. For each globus-objekt, compute its relations to all other globus-objekts. This is done outside CLASSIC in Lisp with the help of CLASSIC functions like cl-concept-instances which returns a list of all instances of a concept.
3. Finally, assert the values computed into the appropriate roles using cl-ind-add-fillers.

Although this method is not so elegant, it has its advantage. The relations have to be computed only once after all the objects have been created. And if the relations have inverse relations such as in the case of orientation relations (south/north and east/west), then only half of the relations have to be computed and asserted. The matching inverse roles are automatically filled by CLASSIC. For example, generating all south fillers for all the objects also fills in the north fillers of all objects. Once the roles have been filled, querying requires no more computation but merely retrieval of the pre-computed values through the cl-fillers function.

Chapter 4

Summary and Further Work

In this work, the conceptual model of the Behaim globe was augmented with qualitative models of topological and orientation relations, information that matters most to people when regarding geographic space[12]. Furthermore, the important notion of scale is considered. Together with a retained quantitative sense of distance, the model was implemented in the description logic system CLASSIC. Although in some ways limited, the resulting system is expressive enough to be able to answer fairly complicated geographic queries.

The Behaim conceptual model has not been fully implemented in CLASSIC. Only the globus-objekt hierarchy with their relevant spatial properties were ported into the CLASSIC system. Implementing the rest should not pose much of a problem. So far, the functions used for computing the topological relations are meant to work for the rectangular regions of the simple example map. No major problems are expected in modifying them to be able to handle the more complex polygons of the Behaim globe. The algorithms needed are described in most computational geometry books.

The model should then be tested with actual data from the Behaim globe. Psychologists and cognitive scientists can be consulted and end-users surveyed to verify, modify and refine the models used for spatial information.

Finally, although the work was done primarily with the Behaim globe in mind, it is not inconcievable to apply the model and methods used here to other maps. Investigating the applicability of integrating such a qualitative model of space to other map types and even sets of maps (from different sources and times to be overlapped and compared) can prove very useful in the design of future intelligent geographic information systems.

Appendix A

Source Code Listing

A.1 (Partial) CLASSIC Implementation of Behaim Globe Conceptual Model Derived from CLOS Model

```
;=====ROLES=====
;globus-root
(cl-define-primitive-role 'name :attribute t)
(cl-define-primitive-role 'urheber :attribute t)

;globus-objekt
(cl-define-primitive-role 'kommentar :attribute t)
(cl-define-primitive-role 'farbe)
(cl-define-primitive-role 'zustand :attribute t)
(cl-define-primitive-role 'erhaltung :attribute t)
(cl-define-primitive-role 'quelle)
(cl-define-primitive-role 'metakommentar :attribute t)
(cl-define-primitive-role 'globus-koord :attribute t)
(cl-define-primitive-role 'globus-polygon :attribute t)
(cl-define-primitive-role 'pixel-polygon :attribute t)
(cl-define-primitive-role 'rasterfeld :attribute t)
(cl-define-primitive-role 'lfdnr :attribute t)
(cl-define-primitive-role 'medienobj)

;globus-region

;nongeo-region
(cl-define-primitive-role 'nongeo-teil-von :attribute t :inverse 'nongeo-teile)

;textregion
(cl-define-primitive-role 'handschrift :attribute t)

;inschrift
(cl-define-primitive-role 'lesung)
(cl-define-primitive-role 'fussnoten :attribute t)
(cl-define-primitive-role 'schreibvari :attribute t)

;geo-region
(cl-define-primitive-role 'moderner-name :attribute t)
(cl-define-primitive-role 'original-name :attribute t)
(cl-define-primitive-role 'moderne-koord :attribute t)
(cl-define-primitive-role 'geo-teil-von :inverse 'geo-teile)
(cl-define-primitive-role 'geo-nachbar :inverse 'geo-nachbar)

;legende
(cl-define-primitive-role 'infotyp)
```

```
;herrschaftszeichen
(cl-define-primitive-role 'herkunft :attribute t)

;=====CONCEPTS=====
(cl-define-primitive-concept 'OBJEKT-NAME 'CLASSIC-THING)

(cl-define-primitive-concept 'HYTEXT 'CLASSIC-THING)

(cl-define-primitive-concept 'FARBE 'CLASSIC-THING)
(cl-populate 'FARBE '(Blau Braun Dunkelbraun Dunkelgrau Einfarbig Gelb Gold Grau Gruen Hellbraun Hellocker
Mehrfarbig Ocker Rosa Rot Rotbraun Schwarz Silber Unbestimmt Weiss))

(cl-define-primitive-concept 'ZUSTAND 'CLASSIC-THING)
(cl-populate 'ZUSTAND '(Abstrakt Fiktiv Illustrativ Mythisch Real Unbestimmt))

(cl-define-primitive-concept 'ERHALTUNGSZUSTAND 'CLASSIC-THING)
(cl-populate 'ERHALTUNGSZUSTAND '(Erkennbar Lesbar Teilweise-erkennbar Teilweise-lesbar Teilweise-verdorben
Unleserlich Verdorben))

(cl-define-primitive-concept 'ZITAT 'CLASSIC-THING)

(cl-define-primitive-concept 'MEDIENOBJEKT 'CLASSIC-THING)

(cl-define-primitive-concept 'TYPOGRAPHIE 'CLASSIC-THING)

(cl-define-primitive-concept 'AUTORSTRING 'CLASSIC-THING)

(cl-define-primitive-concept 'INFOTYP 'CLASSIC-THING)

(cl-define-primitive-concept 'LESUNG 'CLASSIC-THING)

;=====CONCEPT TAXONOMY STARTING WITH GLOBUS-ROOT=====
(cl-define-primitive-concept 'GLOBUS-ROOT '(and CLASSIC-THING
                                                (ALL name OBJEKT-NAME)
                                                (ALL urheber STRING)))

(cl-define-primitive-concept 'GLOBUS-OBJEKT '(and GLOBUS-ROOT
                                                  (ALL kommentar HYTEXT)
                                                  (ALL farbe (ONE-OF Blau Braun Dunkelbraun Dunkelgrau Einfarbig Gelb
Gold Grau Gruen Hellbraun Hellocker Mehrfarbig Ocker Rosa Rot Rotbraun Schwarz Silber Unbestimmt Weiss))
                                                (ALL zustand (ONE-OF Abstrakt Fiktiv Illustrativ Mythisch Real Unbestimmt))
                                                (ALL erhaltung (ONE-OF Erkennbar Lesbar Teilweise-erkennbar Teilweise-lesbar
Teilweise-verdorben Unleserlich Verdorben))
                                                  (ALL quelle ZITAT)
                                                  (ALL metakommentar STRING)
                                                  (ALL globus-koord LIST)
                                                  (ALL globus-polygon LIST)
                                                  (ALL pixel-polygon LIST)
                                                  (ALL rasterfeld LIST)
                                                  (ALL lfdnr INTEGER)
                                                  (ALL medienobj MEDIENOBJEKT)))

(cl-define-primitive-concept 'GLOBUS-REGION 'GLOBUS-OBJEKT)

(cl-define-disjoint-primitive-concept 'NG-REGION '(and GLOBUS-REGION
                                                       (ALL nongeo-teil-von GLOBUS-REGION))
                                      'ng-g-region)
(cl-define-disjoint-primitive-concept 'G-REGION '(and GLOBUS-REGION
                                                      (ALL geo-teil-von GLOBUS-REGION)
                                                      (ALL geo-nachbar GLOBUS-REGION)
                                                      (ALL geo-teile GLOBUS-REGION)
                                                      (ALL nongeo-teile NG-REGION))
                                      'ng-g-region)
(cl-add-rule 'g-region-geo-teil-von @c{G-REGION} '(ALL geo-teil-von G-REGION))
(cl-add-rule 'g-region-geo-nachbar @c{G-REGION} '(ALL geo-nachbar G-REGION))
(cl-add-rule 'g-region-geo-teile @c{G-REGION} '(ALL geo-teile G-REGION))
```

```
(cl-define-disjoint-primitive-concept 'TEXTREGION '(and NG-REGION
                                                        (ALL handschrift TYPOGRAPHIE))
                                      'text-bild-region)
(cl-define-disjoint-primitive-concept 'BILDREGION 'NG-REGION 'text-bild-region)


(cl-define-disjoint-primitive-concept 'ZG-REGION 'G-REGION 'zg-nzg-region)
(cl-define-disjoint-primitive-concept 'NZG-REGION 'G-REGION 'zg-nzg-region)

(cl-define-disjoint-primitive-concept 'LAND-REGION 'G-REGION 'land-wasser-region)
(cl-define-disjoint-primitive-concept 'WASSER-REGION 'G-REGION 'land-wasser-region)

(cl-define-primitive-concept 'ZGR-LAND '(and ZG-REGION
                                             LAND-REGION))

(cl-define-primitive-concept 'ZGR-WASSER '(and ZG-REGION
                                               WASSER-REGION))

(cl-define-primitive-concept 'NZGR-LAND '(and NZG-REGION
                                              LAND-REGION))

(cl-define-primitive-concept 'NZGR-WASSER '(and NZG-REGION
                                                WASSER-REGION))

(cl-define-disjoint-primitive-concept 'INSCHRIFT '(and TEXTREGION
                                                       (ALL lesung AUTORSTRING)
                                                       (ALL fussnoten STRING)
                                                       (ALL schreibvari HYTEXT))
                                      'textregiontype)
(cl-define-disjoint-primitive-concept 'LEGENDE '(and TEXTREGION
                                                     (ALL infotyp INFOTYP)
                                                     (ALL lesung LESUNG))
                                      'textregiontype)
(cl-define-disjoint-primitive-concept 'LANGTEXT '(and TEXTREGION
                                                      (ALL infotyp INFOTYP)
                                                      (ALL lesung LESUNG))
                                      'textregiontype)

(cl-define-disjoint-primitive-concept 'SEGMENT 'BILDREGION 'bildregiontype)
(cl-define-disjoint-primitive-concept 'MINIATUR 'BILDREGION 'bildregiontype)
(cl-define-disjoint-primitive-concept 'KREISLINIE 'BILDREGION 'bildregiontype)

(cl-define-disjoint-primitive-concept 'HERRSCHAFTSZEICHEN '(and MINIATUR
                                                                (ALL herkunft STRING))
                                      'miniaturtype)
(cl-define-disjoint-primitive-concept 'ASTRONOMISCHES-ZEICHEN 'MINIATUR 'miniaturtype)
(cl-define-disjoint-primitive-concept 'TIERZEICHEN 'MINIATUR 'miniaturtype)
(cl-define-disjoint-primitive-concept 'MENSCHENZEICHEN 'MINIATUR 'miniaturtype)
(cl-define-disjoint-primitive-concept 'BAUWERK 'MINIATUR 'miniaturtype)
(cl-define-disjoint-primitive-concept 'SCHIFF 'MINIATUR 'miniaturtype)
(cl-define-disjoint-primitive-concept 'SZENE 'MINIATUR 'miniaturtype)

(cl-define-disjoint-primitive-concept 'FESTLAND 'ZGR-LAND 'zgr-landtype)
(cl-define-disjoint-primitive-concept 'LANDGEBIET 'ZGR-LAND 'zgr-landtype)
(cl-define-disjoint-primitive-concept 'INSEL 'ZGR-LAND 'zgr-landtype)
(cl-define-disjoint-primitive-concept 'STADT 'ZGR-LAND 'zgr-landtype)
(cl-define-disjoint-primitive-concept 'WUESTE 'ZGR-LAND 'zgr-landtype)
(cl-define-disjoint-primitive-concept 'WALD 'ZGR-LAND 'zgr-landtype)
(cl-define-disjoint-primitive-concept 'GEBIRGE 'ZGR-LAND 'zgr-landtype)
(cl-define-disjoint-primitive-concept 'EISFLAECHE 'ZGR-LAND 'zgr-landtype)
(cl-define-disjoint-primitive-concept 'KAP 'ZGR-LAND 'zgr-landtype)

(cl-define-disjoint-primitive-concept 'MEER 'ZGR-WASSER 'zgr-wassertype)
(cl-define-disjoint-primitive-concept 'MEERESGEBIET 'ZGR-WASSER 'zgr-wassertype)
(cl-define-disjoint-primitive-concept 'OZEAN 'ZGR-WASSER 'zgr-wassertype)
```

```
(cl-define-disjoint-primitive-concept 'STROEMUNG 'ZGR-WASSER 'zgr-wassertype)
(cl-define-disjoint-primitive-concept 'SEE 'ZGR-WASSER 'zgr-wassertype)
(cl-define-disjoint-primitive-concept 'FLUSS 'ZGR-WASSER 'zgr-wassertype)

(cl-define-disjoint-primitive-concept 'KONTINENT 'NZGR-LAND 'nzgr-landtype)
(cl-define-disjoint-primitive-concept 'INSELGRUPPE 'NZGR-LAND 'nzgr-landtype)

(cl-define-disjoint-primitive-concept 'SEENPLATTE 'NZGR-WASSER 'nzgr-wassertype)

(cl-define-disjoint-primitive-concept 'FAHNE 'HERRSCHAFTSZEICHEN 'herrschaftszeichentype)
(cl-define-disjoint-primitive-concept 'WAPPEN 'HERRSCHAFTSZEICHEN 'herrschaftszeichentype)
(cl-define-disjoint-primitive-concept 'THRON 'HERRSCHAFTSZEICHEN 'herrschaftszeichentype)
(cl-define-disjoint-primitive-concept 'ZELT 'HERRSCHAFTSZEICHEN 'herrschaftszeichentype)

(cl-define-disjoint-primitive-concept 'TIERKREISZEICHEN 'ASTRONOMISCHES-ZEICHEN 'astronomisches-zeichentype)
(cl-define-disjoint-primitive-concept 'PLANET 'ASTRONOMISCHES-ZEICHEN 'astronomisches-zeichentype)

(cl-define-disjoint-primitive-concept 'LANDTIER 'TIERZEICHEN 'tierzeichentype)
(cl-define-disjoint-primitive-concept 'SEETIER 'TIERZEICHEN 'tierzeichentype)
(cl-define-disjoint-primitive-concept 'FABELWESEN 'TIERZEICHEN 'tierzeichentype)

(cl-define-disjoint-primitive-concept 'HERRSCHER 'MENSCHENZEICHEN 'menschenzeichentype)
(cl-define-disjoint-primitive-concept 'BISCHOF 'MENSCHENZEICHEN 'menschenzeichentype)
(cl-define-disjoint-primitive-concept 'HEILIGER-APOSTEL 'MENSCHENZEICHEN 'menschenzeichentype)
(cl-define-disjoint-primitive-concept 'FIGUR 'MENSCHENZEICHEN 'menschenzeichentype)

(cl-define-disjoint-primitive-concept 'FESTUNG 'BAUWERK 'bauwerktype)
(cl-define-disjoint-primitive-concept 'DENKMAL 'BAUWERK 'bauwerktype)
(cl-define-disjoint-primitive-concept 'GRABMAL 'BAUWERK 'bauwerktype)
(cl-define-disjoint-primitive-concept 'STADTSILHOUETTE 'BAUWERK 'bauwerktype)
```

A.2 Lisp Code

A.2.1 General Functions

```
(defun sqr (x) (* x x))

(defun deg(x) (round (/ (* x 180) PI)))

(defun get-distance (two)
  (sqrt (+ (sqr (- (caar two) (caadr two)))
           (sqr (- (cadar two) (cadadr two)))
)))

 (defun get-globus-koord (ind)
  (let*
      ((globus-koord-role (cl-named-role 'globus-koord))
       (globus-koord (car (cl-fillers ind globus-koord-role))))
    globus-koord
    )
)

 (defun get-globus-polygon (ind)
  (let* ((globus-polygon-role (cl-named-role 'globus-polygon))
         (globus-polygon (car (cl-fillers ind globus-polygon-role))))
    globus-polygon
    )
)

; input: (breite laenge); breite in [-90, 90], laenge in [0, 360]
; output (x y); (laenge breite) laenge in [0, 360], breite in [0 180]
; such that 0,0 is in the lower left corner (quadrant I of cartesian coordinate system)
(defun beh2quad1 (point)
```

```
  (list (cadr point) (+ (car point) 90))
)
```

A.2.2 Topology

```
; returns true if point is inside polygon area (not counting border)
(defun point-in-polygon (point polygon)
  (let* ((pointx (car point))
         (pointy (cadr point))
         (polygonx1 (car (car polygon)))
         (polygony1 (cadr (car polygon)))
         (polygonx3 (car (caddr polygon)))
         (polygony3 (cadr (caddr polygon)))
         )
    (cond ((and (< polygonx1 pointx polygonx3) (< polygony3 pointy polygony1)) t)
          (t nil)
          )
    )
  )

; returns true if point is inside polygon area or on border
(defun point-in-extpolygon (point polygon)
  (let* ((pointx (car point))
         (pointy (cadr point))
         (polygonx1 (car (car polygon)))
         (polygony1 (cadr (car polygon)))
         (polygonx3 (car (caddr polygon)))
         (polygony3 (cadr (caddr polygon)))
         )
    (cond ((and (<= polygonx1 pointx polygonx3) (<= polygony3 pointy polygony1)) t)
          (t nil)
          )
    )
  )

(defun get-rcc (ind obj)
  (let* ((ind-globus-polygon (get-globus-polygon ind))
         (obj-globus-polygon (get-globus-polygon obj))
         (ind-quad1-polygon (mapcar 'beh2quad1 ind-globus-polygon))
         (obj-quad1-polygon (mapcar 'beh2quad1 obj-globus-polygon))
         (ind-pt1 (car ind-quad1-polygon))
         (ind-pt2 (cadr ind-quad1-polygon))
         (ind-pt3 (caddr ind-quad1-polygon))
         (ind-pt4 (cadddr ind-quad1-polygon))
         (obj-pt1 (car obj-quad1-polygon))
         (obj-pt2 (cadr obj-quad1-polygon))
         (obj-pt3 (caddr obj-quad1-polygon))
         (obj-pt4 (caddr obj-quad1-polygon))
         )
    (cond ((equal ind-globus-polygon obj-globus-polygon) 'eq)
          ((and (point-in-polygon obj-pt1 ind-quad1-polygon)
                (point-in-polygon obj-pt3 ind-quad1-polygon)) 'ntpp)
          ((and (point-in-polygon ind-pt1 obj-quad1-polygon)
                (point-in-polygon ind-pt3 obj-quad1-polygon)) 'ntppi)
          ((and (point-in-extpolygon obj-pt1 ind-quad1-polygon)
                (point-in-extpolygon obj-pt3 ind-quad1-polygon)) 'tpp)
          ((and (point-in-extpolygon ind-pt1 obj-quad1-polygon)
                (point-in-extpolygon ind-pt3 obj-quad1-polygon)) 'tppi)
          ((or
            (or (point-in-polygon obj-pt1 ind-quad1-polygon)
                (point-in-polygon obj-pt2 ind-quad1-polygon)
                (point-in-polygon obj-pt3 ind-quad1-polygon)
                (point-in-polygon obj-pt4 ind-quad1-polygon))
            (or (point-in-polygon ind-pt1 obj-quad1-polygon)
```

```
            (point-in-polygon ind-pt2 obj-quad1-polygon)
            (point-in-polygon ind-pt3 obj-quad1-polygon)
            (point-in-polygon ind-pt4 obj-quad1-polygon))) 'po)
        ((or
          (or (point-in-extpolygon obj-pt1 ind-quad1-polygon)
            (point-in-extpolygon obj-pt2 ind-quad1-polygon)
            (point-in-extpolygon obj-pt3 ind-quad1-polygon)
            (point-in-extpolygon obj-pt4 ind-quad1-polygon))
          (or (point-in-extpolygon ind-pt1 obj-quad1-polygon)
            (point-in-extpolygon ind-pt2 obj-quad1-polygon)
            (point-in-extpolygon ind-pt3 obj-quad1-polygon)
            (point-in-extpolygon ind-pt4 obj-quad1-polygon))) 'ec)
        (t 'dc)
        )
))

(defun is-eq? (ind ref)
  (let* ((rel (get-rcc(cl-named-ind ref) ind))
        )
    (cond ((equal rel 'eq) T)
        (T nil)
        )
    )
  )

(defun is-ntpp? (ind ref)
  (let* ((rel (get-rcc(cl-named-ind ref) ind))
        )
    (cond ((equal rel 'ntpp) T)
        (T nil)
        )
    )
  )

(defun is-ntppi? (ind ref)
  (let* ((rel (get-rcc(cl-named-ind ref) ind))
        )
    (cond ((equal rel 'ntppi) T)
        (T nil)
        )
    )
  )

(defun is-tpp? (ind ref)
  (let* ((rel (get-rcc(cl-named-ind ref) ind))
        )
    (cond ((equal rel 'tpp) T)
        (T nil)
        )
    )
  )

(defun is-tppi? (ind ref)
  (let* ((rel (get-rcc(cl-named-ind ref) ind))
        )
    (cond ((equal rel 'tppi) T)
        (T nil)
        )
    )
  )

(defun is-po? (ind ref)
  (let* ((rel (get-rcc(cl-named-ind ref) ind))
        )
    (cond ((equal rel 'po) T)
        (T nil)
```

```
        )
    )
  )

(defun is-ec? (ind ref)
  (let* ((rel (get-rcc(cl-named-ind ref) ind))
         )
    (cond ((equal rel 'ec) T)
          (T nil)
          )
    )
  )

(defun is-dc? (ind ref)
  (let* ((rel (get-rcc(cl-named-ind ref) ind))
         )
    (cond ((equal rel 'dc) T)
          (T nil)
          )
    )
  )

(defun is-inside? (ind ref)
 (cond ((or (is-tpp? ind ref) (is-ntpp? ind ref)) T)
       (T nil)
       )
 )
```

A.2.3 Orientation

```
; returns direction in degrees or -1 if it is the same object (identity)
(defun get-direction (two)
  (setq d (get-distance two))
  (cond ((= d 0) -1)
        (T (setq cost (deg (acos (/ (- (caadr two) (caar two)) d))))
           (setq sint (deg (asin (/ (- (cadadr two) (cadar two)) d))))
           (cond ((= (signum sint) -1) (- 360 cost))
                 (T cost))
           )
        )
)

; returns id, south, north, east or west
(defun my-get-direction (two)
  (let*((twoquad1 (mapcar 'beh2quad1 two))
        (degrees (get-direction twoquad1))
        (innerxdistance (abs (- (caar twoquad1) (caadr twoquad1))))
        (outerxdistance (- 360 innerxdistance))
        )
    (cond
     ((= degrees -1) 'id)
     ((and (> degrees 45) (<= degrees 135)) 'north)
     ((and (> degrees 135) (<= degrees 225))
      (if (< innerxdistance outerxdistance) 'west 'east))
     ((and (> degrees 225) (<= degrees 315)) 'south)
     ((or (and (> degrees 315) (<= degrees 360))
          (and (>= degrees 0) (<=  degrees 45)))
      (if (< innerxdistance outerxdistance) 'east 'west))
     )
    )
  )

(defun lies-in-dir? (ind ref dir)
  (let* ((ind-globus-koord (get-globus-koord ind))
```

```
        (ref-globus-koord (get-globus-koord (cl-named-ind ref)))
        (ind-dir (my-get-direction (list ref-globus-koord ind-globus-koord)))
        )
    (cond ((equal ind-dir dir) T)
          (T nil)
          )
    )
)
```

A.2.4 Distance

```
(defun lies-within-dist? (ind ref dist)
  (let* ((ind-globus-koord (get-globus-koord ind))
         (ref-globus-koord (get-globus-koord (cl-named-ind ref)))
         (d (get-distance (mapcar 'beh2quad1 (list ref-globus-koord ind-globus-koord))))
         )
    (cond ((<= d dist) T)
          (T nil)
          )
    )
)
```

Appendix B

Simple World Example

B.1 A-Box

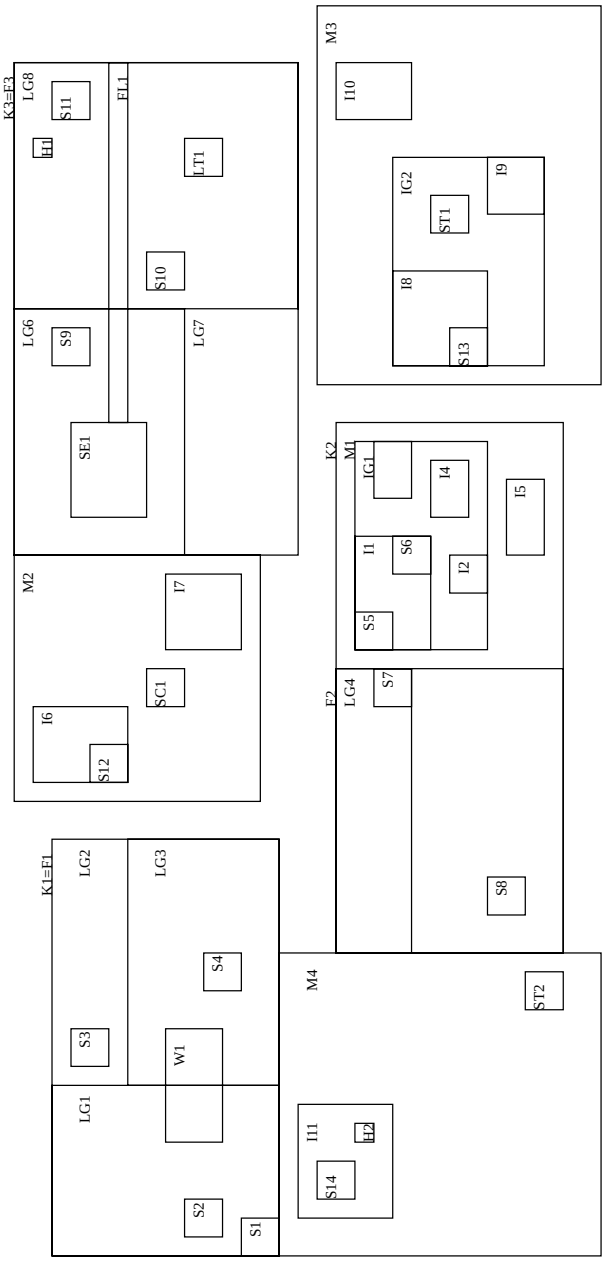
```
(cl-clear-kb)

;kontinent
(cl-create-ind 'k1 '(and KONTINENT
                     (fills globus-koord '(40.0 65.0))
                     (fills globus-polygon '( (70.0 10.0) (70.0 120.0) (10.0 120.0) (10.0 10.0) ))))
(cl-create-ind 'k2 '(and KONTINENT
                    (fills globus-koord '(-35.0 160.0))
                    (fills globus-polygon '( (-5.0 90.0) (-5.0 230.0) (-65.0 230.0) (-65.0 90.0) ))))
(cl-create-ind 'k3 '(and KONTINENT
                     (fills globus-koord '(42.5 260.0))
                     (fills globus-polygon '( (80.0 195.0) (80.0 325.0) (5.0 325.0) (5.0 195.0) ))))

;festland
(cl-create-ind 'f1 '(and FESTLAND
                     (fills globus-koord '(40.0 65.0))
                     (fills globus-polygon '( (70.0 10.0) (70.0 120.0) (10.0 120.0) (10.0 10.0) ))))
(cl-create-ind 'f2 '(and FESTLAND
                     (fills globus-koord '(-35.0 127.5))
                     (fills globus-polygon '( (-5.0 90.0) (-5.0 165.0) (-65.0 165.0) (-65.0 90.0) ))))
(cl-create-ind 'f3 '(and FESTLAND
                     (fills globus-koord '(42.5 260.0))
                     (fills globus-polygon '( (80.0 195.0) (80.0 325.0) (5.0 325.0) (5.0 195.0) ))))

;landgebiet
(cl-create-ind 'lg1 '(and LANDGEBIET
                      (fills globus-koord '(40.0 32.5))
                      (fills globus-polygon '( (70.0 10.0) (70.0 55.0) (10.0 55.0) (10.0 10.0) ))))
(cl-create-ind 'lg2 '(and LANDGEBIET
                     (fills globus-koord '(60.0 87.5))
                     (fills globus-polygon '( (70.0 55.0) (70.0 120.0) (50.0 120.0) (50.0 55.0) ))))
(cl-create-ind 'lg3 '(and LANDGEBIET
                      (fills globus-koord '(30.0 87.5))
                      (fills globus-polygon '( (50.0 55.0) (50.0 120.0) (10.0 120.0) (10.0 55.0) ))))
(cl-create-ind 'lg4 '(and LANDGEBIET
                      (fills globus-koord '(-15.0 127.5))
                      (fills globus-polygon '( (-5.0 90.0) (-5.0 165.0) (-25.0 165.0) (-25.0 90.0) ))))
(cl-create-ind 'lg5 '(and LANDGEBIET
                      (fills globus-koord '(-45.0 127.5))
                      (fills globus-polygon '( (-25.0 90.0) (-25.0 165.0) (-65.0 165.0) (-65.0 90.0) ))))
(cl-create-ind 'lg6 '(and LANDGEBIET
                      (fills globus-koord '(57.5 227.5))
                      (fills globus-polygon '( (80.0 195.0) (80.0 260.0) (35.0 260.0) (35.0 195.0) ))))
(cl-create-ind 'lg7 '(and LANDGEBIET
```

```
                    (fills globus-koord '(20.0 227.5))
                    (fills globus-polygon '( (35.0 195.0) (35.0 260.0) (5.0 260.0) (5.0 195.0) ))))
(cl-create-ind 'lg8 '(and LANDGEBIET
                    (fills globus-koord '(42.5 292.5))
                    (fills globus-polygon '( (80.0 260.0) (80.0 325.0) (5.0 325.0) (5.0 260.0) ))))

;inselgruppe
(cl-create-ind 'ig1 '(and INSELGRUPPE
                    (fills globus-koord '(-27.5 197.5))
                    (fills globus-polygon '( (-10.0 170.0) (-10.0 225.0) (-45.0 225.0) (-45.0 170.0) ))))
(cl-create-ind 'ig2 '(and INSELGRUPPE
                    (fills globus-koord '(-40.0 272.5))
                    (fills globus-polygon '( (-20.0 245.0) (-20.0 300.0) (-60.0 300.0) (-60.0 245.0) ))))

;insel
(cl-create-ind 'i1 '(and INSEL
                    (fills globus-koord '(-20.0 185.0))
                    (fills globus-polygon '( (-10.0 170.0) (-10.0 200.0) (-30.0 200.0) (-30.0 170.0) ))))
(cl-create-ind 'i2 '(and INSEL
                    (fills globus-koord '(-40.0 190.0))
                    (fills globus-polygon '( (-35.0 185.0) (-35.0 195.0) (-45.0 195.0) (-45.0 185.0) ))))
(cl-create-ind 'i3 '(and INSEL
                    (fills globus-koord '(-20.0 217.5))
                    (fills globus-polygon '( (-15.0 210.0) (-15.0 225.0) (-25.0 225.0) (-25.0 210.0) ))))
(cl-create-ind 'i4 '(and INSEL
                    (fills globus-koord '(-35.0 212.5))
                    (fills globus-polygon '( (-30.0 205.0) (-30.0 220.0) (-40.0 220.0) (-40.0 205.0) ))))
(cl-create-ind 'i5 '(and INSEL
                    (fills globus-koord '(-55.0 205.0))
                    (fills globus-polygon '( (-50.0 195.0) (-50.0 215.0) (-60.0 215.0) (-60.0 195.0) ))))
(cl-create-ind 'i6 '(and INSEL
                    (fills globus-koord '(62.5 145.0))
                    (fills globus-polygon '( (75.0 135.0) (75.0 155.0) (50.0 155.0) (50.0 135.0) ))))
(cl-create-ind 'i7 '(and INSEL
                    (fills globus-koord '(30.0 180.0))
                    (fills globus-polygon '( (40.0 170.0) (40.0 190.0) (20.0 190.0) (20.0 170.0) ))))
(cl-create-ind 'i8 '(and INSEL
                    (fills globus-koord '(-32.5 257.5))
                    (fills globus-polygon '( (-20.0 245.0) (-20.0 270.0) (-45.0 270.0) (-45.0 245.0) ))))
(cl-create-ind 'i9 '(and INSEL
                    (fills globus-koord '(-52.5 292.5))
                    (fills globus-polygon '( (-45.0 285.0) (-45.0 300.0) (-60.0 300.0) (-60.0 285.0) ))))
(cl-create-ind 'i10 '(and INSEL
                     (fills globus-koord '(-15.0 317.5))
                     (fills globus-polygon '( (-5.0 310.0) (-5.0 325.0) (-25.0 325.0) (-25.0 310.0) ))))
(cl-create-ind 'i11 '(and INSEL
                     (fills globus-koord '(-7.5 35.0))
                     (fills globus-polygon '( (5.0 20.0) (5.0 50.0) (-20.0 50.0) (-20.0 20.0) ))))

;stadt
(cl-create-ind 's1 '(and STADT
                    (fills globus-koord '(15.0 15.0))
                    (fills globus-polygon '( (20.0 10.0) (20.0 20.0) (10.0 20.0) (10.0 10.0) ))))
(cl-create-ind 's2 '(and STADT
                    (fills globus-koord '(30.0 20.0))
                    (fills globus-polygon '( (35.0 15.0) (35.0 25.0) (25.0 25.0) (25.0 15.0) ))))
(cl-create-ind 's3 '(and STADT
                    (fills globus-koord '(60.0 65.0))
                    (fills globus-polygon '( (65.0 60.0) (65.0 70.0) (55.0 70.0) (55.0 60.0) ))))
(cl-create-ind 's4 '(and STADT
                    (fills globus-koord '(25.0 85.0))
                    (fills globus-polygon '( (30.0 80.0) (30.0 90.0) (20.0 90.0) (20.0 80.0) ))))
(cl-create-ind 's5 '(and STADT
                    (fills globus-koord '(-15.0 175.0))
                    (fills globus-polygon '( (-10.0 170.0) (-10.0 180.0) (-20.0 180.0) (-20.0 170.0) ))))
(cl-create-ind 's6 '(and STADT
```

```
                    (fills globus-koord '(-25.0 195.0))
                    (fills globus-polygon '( (-20.0 190.0) (-20.0 200.0) (-30.0 200.0) (-30.0 190.0) ))))
(cl-create-ind 's7 '(and STADT
                    (fills globus-koord '(-20.0 160.0))
                    (fills globus-polygon '( (-15.0 155.0) (-15.0 165.0) (-25.0 165.0) (-25.0 155.0) ))))
(cl-create-ind 's8 '(and STADT
                    (fills globus-koord '(-50.0 105.0))
                    (fills globus-polygon '( (-45.0 100.0) (-45.0 110.0) (-55.0 110.0) (-55.0 100.0) ))))
(cl-create-ind 's9 '(and STADT
                    (fills globus-koord '(65.0 250.0))
                    (fills globus-polygon '( (70.0 245.0) (70.0 255.0) (60.0 255.0) (60.0 245.0) ))))
(cl-create-ind 's10 '(and STADT
                     (fills globus-koord '(40.0 275.0))
                     (fills globus-polygon '( (45.0 270.0) (45.0 280.0) (35.0 280.0) (35.0 270.0) ))))
(cl-create-ind 's11 '(and STADT
                     (fills globus-koord '(65.0 315.0))
                     (fills globus-polygon '( (70.0 310.0) (70.0 320.0) (60.0 320.0) (60.0 310.0) ))))
(cl-create-ind 's12 '(and STADT
                     (fills globus-koord '(55.0 140.0))
                     (fills globus-polygon '( (60.0 135.0) (60.0 145.0) (50.0 145.0) (50.0 135.0) ))))
(cl-create-ind 's13 '(and STADT
                     (fills globus-koord '(-40.0 250.0))
                     (fills globus-polygon '( (-35.0 245.0) (-35.0 255.0) (-45.0 255.0) (-45.0 245.0) ))))
(cl-create-ind 's14 '(and STADT
                     (fills globus-koord '(-5.0 30.0))
                     (fills globus-polygon '( (0.0 25.0) (0.0 35.0) (-10.0 35.0) (-10.0 25.0) ))))

;wald
(cl-create-ind 'w1 '(and WALD
                    (fills globus-koord '(32.5 55.0))
                    (fills globus-polygon '( (40.0 40.0) (40.0 70.0) (25.0 70.0) (25.0 40.0) ))))

;see
(cl-create-ind 'se1 '(and SEE
                     (fills globus-koord '(55.0 217.5))
                     (fills globus-polygon '( (65.0 205.0) (65.0 230.0) (45.0 230.0) (45.0 205.0) ))))

;fluss
(cl-create-ind 'fl1 '(and FLUSS
                    (fills globus-koord '(55.0 277.5))
                    (fills globus-polygon '( (55.0 230.0) (55.0 325.0) (55.0 325.0) (55.0 230.0) ))))

;meer
(cl-create-ind 'm1 '(and MEER
                    (fills globus-koord '(-35.0 197.5))
                    (fills globus-polygon '( (-5.0 165.0) (-5.0 230.0) (-65.0 230.0) (-65.0 165.0) ))))
(cl-create-ind 'm2 '(and MEER
                    (fills globus-koord '(47.5 162.5))
                    (fills globus-polygon '( (80.0 130.0) (80.0 195.0) (15.0 195.0) (15.0 130.0) ))))
(cl-create-ind 'm3 '(and MEER
                    (fills globus-koord '(-37.5 290.0))
                    (fills globus-polygon '( (0.0 240.0) (0.0 340.0) (-75.0 340.0) (-75.0 240.0) ))))
(cl-create-ind 'm4 '(and MEER
                    (fills globus-koord '(-32.5 50.0))
                    (fills globus-polygon '( (10.0 10.0) (10.0 90.0) (-75.0 90.0) (-75.0 10.0) ))))

;landtier
(cl-create-ind 'lt1 '(and LANDTIER
                     (fills globus-koord '(30.0 300.0))
                     (fills globus-polygon '( (35.0 295.0) (35.0 305.0) (25.0 305.0) (25.0 295.0) ))))

;seetier
(cl-create-ind 'st1 '(and SEETIER
                     (fills globus-koord '(-35.0 285.0))
                     (fills globus-polygon '( (-30.0 280.0) (-30.0 290.0) (-40.0 290.0) (-40.0 280.0) ))))
(cl-create-ind 'st2 '(and SEETIER
```

```
                        (fills globus-koord '(-60.0 80.0))
                        (fills globus-polygon '( (-55.0 75.0) (-55.0 85.0) (-65.0 85.0) (-65.0 75.0) ))))

;schiff
(cl-create-ind 'sc1 '(and SCHIFF
                        (fills globus-koord '(40.0 160.0))
                        (fills globus-polygon '( (45.0 155.0) (45.0 165.0) (35.0 165.0) (35.0 155.0) ))))

;herrscher
(cl-create-ind 'h1 '(and HERRSCHER
                        (fills globus-koord '(72.5 302.5))
                        (fills globus-polygon '( (75.0 300.0) (75.0 305.0) (70.0 305.0) (70.0 300.0) ))))
(cl-create-ind 'h2 '(and HERRSCHER
                        (fills globus-koord '(-12.5 42.5))
                        (fills globus-polygon '( (-10.0 40.0) (-10.0 45.0) (-15.0 45.0) (-15.0 40.0) ))))
```

B.2 Map


K3=F3
LG8
S11
EL1
LT1
S10
LG6
S9
LG7
SE1
M2
I7
SC1
I6
S12
M3
I10
IG2
I9
ST1
I8
S13
K2
M1
IG1
I4
I5
I1
S6
I2
S5
E2
LG4
S7
S8
K1=F1
LG2
LG3
S4
S3
W1
LG1
S2
S1
M4
ST2
I11
L2
S14

Bibliography

[1] Bennett, B., Logical Representations for Automated Reasoning about Spatial Relationships, The University of Leeds, School of Computer Studies, PhD Thesis, October 1997.

[2] Borgida, A., et al., CLASSIC: A Structural Data Model for Objects, In: Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data , pp. 59-67, June 1989.

[3] Brachman, R., et al., Living with CLASSIC: When and How to Use a KL-ONE-Like Language, In: Sowa, J., editor, Principles of Semantic Networks: Explorations in the representation of knowledge , Morgan-Kaufmann: San Mateo, California, pp. 401–456, 1991.

[4] Clementini, E., Di Felice, P. and Hernández, D., Qualitative Representation of Positional Information, In: Artificial Intelligence 95(2): pp. 317-356, 1997.

[5] Cohn, A., Calculi for Qualitative Spatial Reasoning, In: Calmet, J. and Campbell, J., editors, Artificial Intelligence and Symbolic Mathematical Computation, LNCS 1138, Springer Verlag, pp. 124-143, 1996.

[6] Cohn, A., et al., Representing and Reasoning with Qualitative Spatial Relations.In: Stock, O., editor, Spatial and Temporal Reasoning, Dordrecht: Kluwer, pp. 97-134, 1997.

[7] Cohn, A., Randell D. and Cui, Z., Taxonomies of Logically Defined Qualitative Spatial Relations, In: International Journal of Human-Computing, special issue on Formal Ontology in Conceptual Analysis and Knowledge Representation (guest editors: N. Guarino and R. Poli), vol 43, Issue 5-6, pp 831-846, 1995.

[8] De Floriani, L., Marzano, P. and Puppo, E., Spatial Queries and Data Models, In: Frank, A. and Campari, I., editors, Spatial Information Theory - A Theoretical Basis for GIS, Lecture Notes in Computer Science, N.716, Springer-Verlag, 1993.

[9] De Floriani, L., Puppo, E. and Magillo, P., Applications of computational geometry to geographic information systems, In: Sack, J. and Urrutia, J., editors, Handbook of Computational Geometry, Elsevier Science, pp.333-388, 1999.

[10] Donini, F., Lenzerini, M., Nardi, D., Schaerf, A.,Reasoning in Description Logics, In: Brewka, G., Principles of Knowledge Representation and Reasoning, Studies in Logic, Language and Information, CLSI Publications, pp. 193-238, 1996.

[11] Dorffner, L., Der digitale Behaim-Globus - Visualisierung und Vermessung des historisch wertvollen Originals, In: Cartographica Helvetica - Fachzeitschrift für Kartengeschichte. Nr. 14/1996, pp. 20-24.

[12] Egenhofer, M. and Mark D., Naive Geography, In: Frank, A. and Kuhn, W., editors, Lecture Notes in Computer Science, Vol. 988, Springer-Verlag, pp. 1-15, September 1995.

[13] Egenhofer, et al., Progress in Computational Methods for Representing Geographic Concepts, In: International Journal of Geographical Information Science 13 (8): pp. 775-796, 1999.

[14] Frank, A., Spatial Ontology, In: Stock, O.,editor, Spatial and Temporal Reasoning, Dordrecht: Kluwer, pp. 135-153, 1997.

[15] Freundschuh, S. and Egenhofer, M., Human Conceptions of Spaces: Implications for GIS, In: Transactions in GIS, 2 (4): pp. 361-375, 1997.

[16] Görz, G., Integrating Knowledge Representation into a Multimedia Information System In: Proceedings of Informatique'95: Interface to Real and Virtual Worlds, Montpellier, 1995.

[17] Görz, G., Holst, N., The Digital Behaim Globe (1492), In: Bearman, D. and Trant, J., editors, MuseumInteractive Multimedia 1997: Cultural Heritage Systems – Design and Interfaces. Selected Papers from ICHIM 97, Fourth International Conference on Hypermedia and Interactivity in Museums, Paris 1997, Pittsburgh, Penn.: Archive & Museum Informatics, pp. 157-173, 1997.

[18] Graham, P., ANSI Common LISP, New Jersey: Prentice Hall, Inc., 1996.

[19] Haarslev, V., Lutz C. and Möller, R., A Description Logic with Concrete Domains and a Role-forming Predicate Operator, In: Journal of Logic and Computation, Vol. 9, No.3, pp. 351-384, June 1999.

[20] Haarslev, V., Lutz C. and Möller, R., Foundations of Spatioterminological Reasoning with Description Logics, In: Cohn, A., Schubert, L. and Shapiro, S., editors, Principles of Knowledge Representation and Reasoning: Proceedings of the Sixth International Conference (KR'98), Morgan-Kaufmann Publishers, pp. 112-123, 1998.

[21] Jelinek, R., Raeumliches Schliessen in einer kartographischen Datenbasis, Universitaet Erlangen-Nuernberg, IMMD IX, Diplomarbeit, Erlangen, August 1997.

[22] Jennings, G., Modern Geometry with Applications, New York: Springer Verlag, 1994.

[23] Jones, C., Geographical Interfaces to Museum Collections, In: Proc. 4th International Conference on Hypermedia & Interactivity in Museums(ICHIM '97), pp. 226-236, Sept 1997.

[24] Larson, R., Geographic Information Retrieval and Spatial Browsing, In: Smith, L. and Gluck, Myke, editors, GIS and Libraries: Patrons, Maps and Spatial Information, Urbana-Champaign : University of Illinois, pp. 81-124, 1996.

[25] Laszlo, M., Computational Geometry and Computer Graphics in C++, New Jersey, Prentice Hall, Inc., 1996.

[26] Laure, V., Spatial Representation and Reasoning in Artificial Intelligence, In: Stock, O., editor, Spatial and Temporal Reasoning, Dordrecht: Kluwer, pp. 5-41, 1997.

[27] Lutz C., Repräsentation topologische Informationen in Beschreibungslogiken, Universitaet Hamburg, Fachbereich Informatik, Diplomarbeit, 1998.

[28] McGuinness, D., Resnick, L. and Isbell C., Description Logic in Practice: ACLASSIC Application, In: Proceedings of the 1995 International Joint Conference on Artificial Intelligence, August 1995.

[29] Möller, R., Neumann, B. and Wessel, M., Towards Computer Vision with Description Logics: Some Recent Progress, In: Proceedings Integration of Speech and Image Understanding, Corfu, Greece, IEEE Computer Society, Los Alamitos, pp. 101–115, 1999.

[30] Möller, R. and Wessel, M., Terminological Default Reasoning about Spatial Information: A First Step, In: Proc. of COSIT'99, International Conference on Spatial Information Theory, Stade, 1999, Springer-Verlag, pp. 189-204, 1999.

[31] Mukerjee, A. and Hernandez, D., Representation of Spatial Knowledge, Tutorial Notes from IJCAI-95, Montreal, August 20, 1995.

[32] Pfannnenstein, J, Wissensmodellierung am Beispiel des Behaim-Globus, Universitaet Erlangen-Nuernberg, IMMD IX, Studienarbeit, Erlangen, September 1993.

[33] Pratt, I., Map Semantics, In: Spatial Information Theory: a theoretical basis for GIS, Lecture Notes in Computer Science volume 716, pp. 77 - 91, Springer-Verlag, Berlin, 1993.

[34] Reiter R. and Mackworth, A., A Logical Framework for Depiction and Image Interpretation, In: Artificial Intelligence 41(2): pp. 125-155, 1989.

[35] Resnick, L., et al., CLASSIC Description and Reference Manual for the COMMON LISP Implementation Version 2.3, AT&T Laboratories, December 1995.

[36] Sharma, J., Flewelling, D. and Egenhofer, M., A Qualitative Spatial Reasoner, In: Sixth International Symposium on Spatial Data Handling, Edinburgh, Scotland, pp. 665-681, September 1994.

[37] Stock, O., editor, Spatial and Temporal Reasoning, Dordrecht: Kluwer, 1997.

[38] Wilensky, R., LISPcraft, New York: W. W. Norton and Company, Inc., 1984.